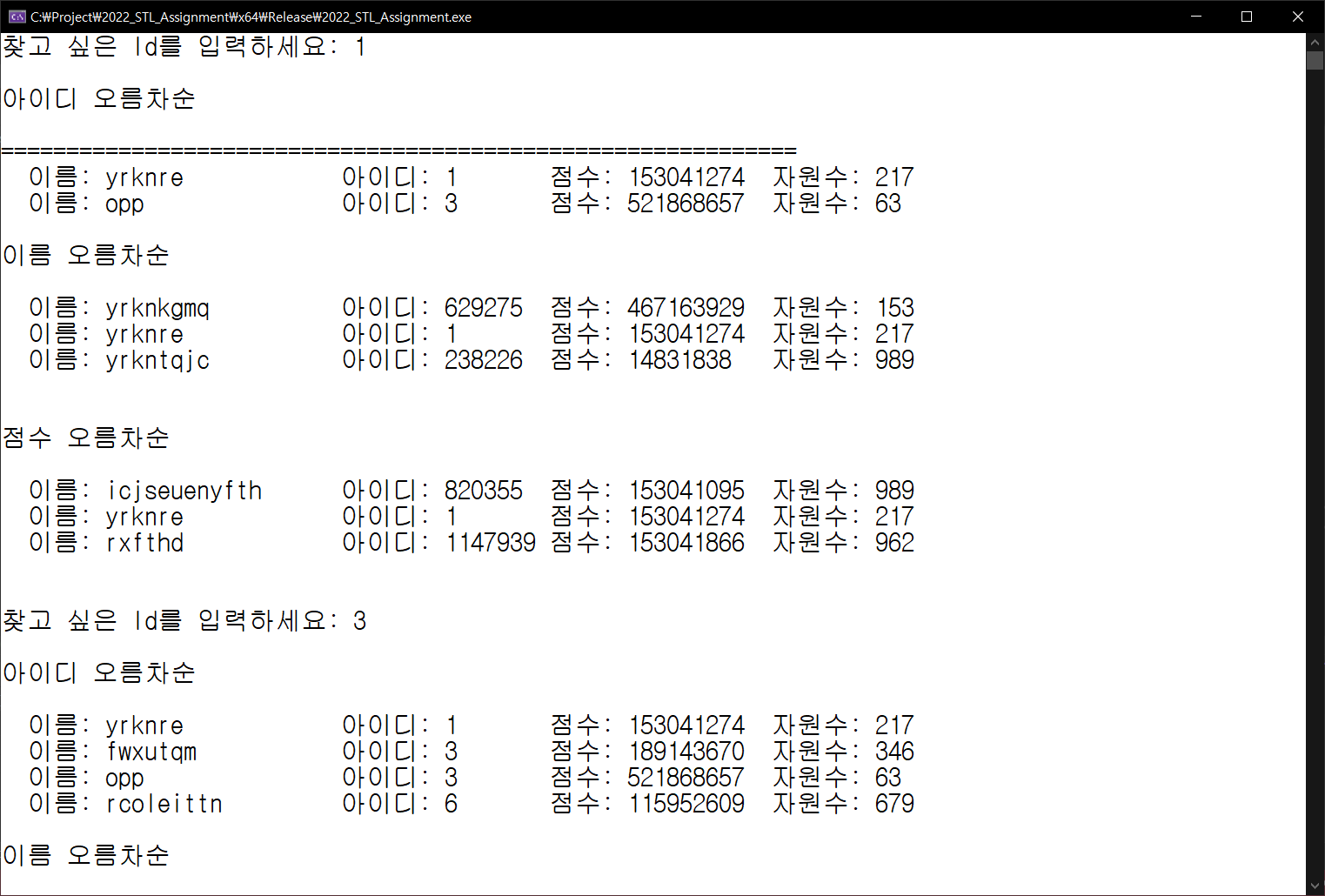
**STL 과제 보고서**

과목명: STL  
학번: 2020182048  
이름: 김정학

**과제 해결 방법**

일단 과제pdf 파일을 열어보니 교수님께서 Player 2백만개의 정보를 binary 모드로 기록하였다고 작성하셨다. Player 클래스의 정의만으로는 파일이 실제로 어떻게 작성되는지 명확하지 않아서 비주얼 스튜디오로 “2022 STL 과제 파일”을 열어보았다. 파일에는 이진법(비트 단위)로 player의 정보가 쓰여 있었고 비주얼 스튜디오는 비트들을 바이트 단위로 읽어 16진법으로 표현해주었다. 또한 쓰여 있는 바이트들이 표현 가능한 아스키코드면, 영문자나 특수문자 등으로 변환도 해주는데, 이를 통해서 Player 객체의 정보에 이어 num만큼의 정보가 바이트 단위로 작성되어 있는 것을 확인했다. 따라서 파일에 쓰여진 플레이어를 차례대로 읽기 위해서는 player의 정보를 읽고 그 player의 num만큼의 바이트를 읽어야 한다고 추정했다.  
 이 생각을 실제 코드로 옮겨 정말로 200만개만큼 읽어지는지 테스트를 해보았는데, 이상하게 잘 읽어져서 중단점을 걸고 비주얼 스튜디오 디버깅 툴을 사용해 읽어온 데이터를 확인해 보았는데, 아무것도 읽히지 않았다. 순간 떠오른 생각으로, 파일이 정상적으로 열렸는지 확인을 해보아야 한다는 것이었다. 파일이 제대로 열렸는지 확인해보았더니, 아니나 다를까 파일이 제대로 열리지 않았다는 것이 판명이 되었다. 교수님께서 과제와 함께 제공하신 실행파일을 통해 만들어진 “2022 STL 과제 파일”이 소스 파일과 같은 폴더 내에 있지 않았기 때문에(경로가 달라서) 읽지 못한 것이었다. 따라서 과제 파일을 소스 파일과 동일한 경로에 두고 프로그램을 실행시켰는데, 작동하자마자 boundary exception이 발생하였다. Player의 num만큼 바이트를 읽어 오는 과정에서 char\*에는 num만큼의 바이트를 저장할 수가 없다고 구글이 친절하게 알려주었다. 이를 해결하기 위해서 num만큼의 바이트를 동적할당 하고 할당된 메모리의 시작주소를 player의 p에 저장하는 방법을 통해 파일을 계속 읽어 나갔다. 결과적으로 파일을 끝까지 읽는데 성공하였고, 실제로 파일에는 200만개의 player의 정보들이 적혀 있음을 알 수 있었다. 여기서 궁금증이 생겨서 199만9999개만큼만 읽게 반복문의 구조를 바꾸면 player가 한 개 적게 읽어졌고, 200만1개를 읽으니, 예상과는 다르게 어떠한 예외도 없이 프로그램이 정상 작동 및 종료되었다. 이유는 파일을 200백만개만큼 읽는 과정에서, 파일의 끝까지 정상적으로 읽는지 확인하기 위해서 while 반복문의 조건식으로 ifstream으로 연 파일을 체크했는데, 이렇게 하면 파일의 어떤 부분을 가리키는 커서가 파일 그 자체의 끝을 가리키는 것이 아니라 파일의 가장 끝 부분을 가리키기 때문에 마지막 객체를 다시 읽는 버그가 발생하게 된다. 그래서 여러가지 시도 끝에 교수님께 메일을 보내 질문을 드렸고, 파일 스트림의 read 함수를 조건으로 체크를 할 수 있다고 수업시간에 배운 내용을 상기시켜 주셨다. 자료형으로는 STL 컨테이너인 vector에 읽은 player 데이터를 저장했지만, player 객체는 추가되거나 삭제되지 않는다는 말이 array를 사용해도 된다는 것임을 나중에 알게 된 후에, vector를 array로 바꾸었고, 결과적으로 player 200만개를 파일에서 잘 읽어왔다. 파일을 읽은 후에는 대부분이 STL 알고리즘 함수들과 컨테이너 내장 함수 등으로 해결이 가능한 부분으로 생각했다.

* 문제 1번: array 컨테이너의 마지막 원소를 반환하는 array 내장 함수 back을 사용하여 마지막 player를 가져온 후 player 클래스에 내장된 show 함수를 사용하여 그 객체의 정보를 출력하는 방법을 사용하여 해결했다.  
  출력결과에 해당하는 player는 이름이 zlzsdt, 아이디가 1785636, 점수는 210159520, 그리고 자원수는 582이다.
* 문제 2번: array에 담겨있는 모든 Player들의 점수를 표준 템플릿 라이브러리인 numeric에 있는 accumulate함수를 사용하여 합계를 long long int에 저장해서 나누는 식으로 했는데, 다른 사람들과 의견을 나누어 보니 총합을 객체의 개수로 나누는 과정에서 소수점 아래 자리의 숫자가 생길 수도 있다고 생각이 들었다. 따라서 플레이어들의 점수 총합을 long long int 자료형에 담고, 나누는 과정에서 double로 형 변환을 한 후 객체 개수로 나누어서 출력을 하였다.  
  평균점수는 300073368.779400이다.
* 문제 3번: 우선 파일 이름이 “바이트 수 999”인 파일에 데이터를 작성해야 하기 때문에 ifstream으로 파일을 binary 모드로 열고, 조건에 맞는 객체의 수를 알아내기 위해 STL 알고리즘 함수 중 하나인 count\_if를 사용하였다. Count\_if는 선형 알고리즘으로, 모든 원소들을 비교하여 num이 999인 player들의 개수를 세는데 비교식으로 사용한 람다 함수에서 교수님께서 작성하신 player 클래스 멤버 함수인 write로 “바이트 수 999”라는 파일에 binary 모드로 작성도 같이 하였다. 이렇게 한 번에 쓰는 것은 컴퓨터는 물론 프로그래머에게 좋은 일이다. 해당 조건에 맞는 player의 개수는 2024개, 작성된 파일의 크기는 2,135,320바이트라고 파일의 속성에는 나와있다. 혹시나 하는 마음에 “바이트 수 999”파일을 읽어서 개수를 확인해보았더니, 2023개로 출력이 되었다. 읽는 부분에도 쓰는 부분에도 문제가 없어서 무엇 때문에 2023개 밖에 작성되지 않는 것인지 특정하는데 시간이 조금 걸렸다. 프로그램이 비정상적으로 종료되면 파일의 작성이 완전히 마무리되지 않기 때문에 발생하는 문제였다. 4번 문제를 해결하기 위한 무한루프에서 제대로 탈출할 수 없어, “바이트 수 999” 파일이 정상적으로 닫히지 않아 마지막 객체가 작성이 안되었다. 따라서 이를 해결하기 위해 문제 3번의 해결 바로 밑에 파일을 닫아주는 작업을 따로 해주었다.
* 4번 문제에서는 만족사항 중 4번 과정을 무한히 반복할 수 있어야 한다는 조건이 있어, while반복문 내에서 아이디 입력을 받고 그 입력이 유효하지 않을 경우에 대한 예외처리를 해주었다. 반복문의 조건으로 cin을 사용할 경우, 올바르지 않은 입력이 들어오면 바로 반복문을 빠져나가기 때문에 무한히 반복할 수 있다는 말과는 조금 거리가 있다고 생각했기 때문이다. 정상적인 입력이 들어오고 해당 id에 맞는 플레이어 객체를 찾는 방법으로 처음에는 find를 사용하였지만, 교수님의 말씀과 다른 학우들과 의견을 나누어 본 끝에 선형 알고리즘인 find 대신에 이진탐색을 사용하는 알고리즘의 경우가 더 효율적인(시간 복잡도가 낮은)방법이라고 생각했다. 하지만 이진탐색 알고리즘을 사용하기 위해서는 정렬이 보장되어야 한다는 전제조건이 있는데, 커다란 문제가 있었다. 현재 내가 작성한 프로그램의 경우에는 정렬전에 find를 통해 찾은 후, 없다면 정렬하지 않고 반복문의 처음으로 바로 돌아가는 방법을 택했는데, 이진탐색을 하려면 정상적인 입력인지 검사한 후 먼저 정렬을 하고 이진탐색을 진행하는데, 없다면 반복문을 다시 돌리면서 이미 정렬된 자료들을 다시 정렬하게 되면서 sort 함수의 시간 복잡도가 n\*log(n)에서 n^2로 바뀌는 마법 같은 일이 발생한다. 아무리 고민해도 이 문제는 따로 해결할 방법이 없는 것 같아서 id순으로 정렬되었는지 아닌지 판단하는 bool 변수를 만들어서 이미 id순으로 정렬된 자료들을 다시 정렬하지 않게 하였다.  
  

**제출한 문제해결 방법이** **효율적이라고 주장하는 이유**

우선 객체의 개수가 변하지 않는다는 말이 명시적으로 써져 있기 때문에, vector보단 array를 사용하였다. 2번은 모든 객체의 점수를 더해야 하기 때문에 선형 알고리즘인 accumulate를 사용했고, 3번에서도 player의 num 순으로 정렬이 되어 있다는 보장을 할 수 없기 때문에 선형 알고리즘인 count\_if를 사용하여 바이트 수가 999인 객체들을 찾아 그 수의 합을 구하고 파일에 작성하였다. 코드의 짜임새가 가장 다르게 되는 부분은 문제 4번이라고 생각했는데, 원래 여기서는 선형 알고리즘인 find를 사용하여 시간 복잡도가 O(n)에 해당하였지만, 최악의 경우 id를 여러 번 입력하고 찾지 못하는 경우에는 비효율적인 코드가 될 것으로 예상이 되었다. 결국 입력 받은 id에 맞는 객체를 찾으면 id순으로 sort를 진행해야 하기 때문에 id순 정렬이 되어있는지 아닌지를 판단하여, 정렬 후에는 전체 객체 중에 해당 id에 맞는 객체가 없어, 다시 찾는 일이 발생하더라도 이진탐색 알고리즘을 이용하여 빠르게 찾을 수 있는 방법을 사용하였다. 또한 sort를 사용할 때 id순으로 정렬을 한 후 이진탐색 알고리즘을 사용하는데, id순으로 sort를 한 후 입력된 id에 해당하는 객체를 찾지 못하면 다음 루프로 넘어가는데, 이 경우에는 이미 정렬된 array를 다시 sort해서 시간 복잡도가 n^2이 되는 마법 같은 일이 일어난다. 따라서 이를 방지하기 위해 id순으로 정렬되었는지를 나타내는 bool 변수를 하나 사용해서 이미 id순으로 정렬된 데이터를 다시 정렬하지 않게 하였다.

**개선할 부분**

위의 문제 4번이 효율적이라고 생각하는 부분에서 이진탐색을 사용하면서도 id순으로 정렬하는 일을 다시 반복하여 sort 함수가 가질 수 있는 최악의 시간 복잡도인 n^2을 가지지 않게 bool 변수로 체크하는 방법을 택했는데, 함수들의 순서를 바꾸거나 다른 방법이 존재할 것만 같은데, 현재의 나로서는 더 나은 방법을 알 수가 없어 지금의 방법으로 코드를 작성하였지만 이를 개선하고 싶다. 뭔가 마음에 안 든다. 또한, 4번에서 입력 받은 값을 체크해서 올바른 값만 입력 받게 하였는데, 이렇게 하면 컨트롤+Z를 입력해서 반복문을 탈출하고, 프로그램을 정상 종료하게 하는 방법이 사라지기 때문에 3번에서 파일이 제대로 닫히지 않아 마지막 객체가 작성되지 않는 방법이 발생했고, 이를 해결하기 위해서 “손수” 파일을 닫아주는 RAII에 어긋나는 방법 또한 개선해야 할 점이라고 생각한다.

**느낀 점**

과제를 하면서 처음에 생각했던 방법보다 더 효율적인 방법도 존재하고, STL 알고리즘들 중 오류가 발생하면 바로바로 작동하게끔 수정하거나 함수들을 추가하는데 어려움을 많이 겪기도 했다. 아직까지 공부해야 할 것들도 많고, 이미 공부하고 지나왔지만 확실치 않은 부분들도 많았다. 예를 들어, 시간 복잡도 등을 고려하지 않은 채 반복문에서 함수를 사용한 부분들이다. 무한히 반복하는 만큼 시간 복잡도를 고려해서 최단 시간에 작동하는 프로그램을 작성해야 했는데, 익숙하고 아는 부분만을 고려해 프로그램을 작성을 해서 차후에 코드를 수정하는 일이 빈번했다. 이미 알고 있는 부분만이 지식의 끝이 아니라는 생각을 하고 늘 공부하는 자세를 가져야 하겠다고 다짐하는 계기가 되었다.  
그리고 STL 과목에 대해서 의논할 친구들이 현재 5명정도 있는데, 다른 사람과 의견을 나누는 것은 항상 옳다는 것을 다시 느꼈다. 공자의 말을 빌리자면 사람 3명이 모이면 반드시 배울 점이 있다고, 역시 여러 사람들의 의견을 듣고 생각하는 것은 늘 발전하는 길로 이어진다고 다시 한번 생각했다.